ASU

BASIC PROPERTIES AND LIMITS OF INTEGRATED ARRAYS OF DISSIPATIVE CIRCUIT AND LOGIC ELEMENTS

Final Technical Report for Contract N00014-85-K-0387

Reported to the Office of Naval Research

Reported by Robert O. Grondin
Department of Electrical Engineering
College of Engineering & Applied Sciences
Arizona State University
Tempe, Arizona 85287-5706

CRR-90030

I. Introduction

This is the final technical report of the progress made on award No. N00014-85-K-0387, a program of research entitled "Basic Properties and Limits of Integrated Arrays of Dissipative Circuit and Logic Elements". The principal investigator was Dr. Robert O. Grondin, an Associate Professor of Electrical and Computer Engineering at Arizona State University.

For reporting purposes we will divide the efforts on this program into the following categories:

(a) the effect of scaling on the performance of integrated device arrays;

(b) the development of a pipelined content addressable memory;

(c) the development of integratable fuzzy neurons;

(d) the mapping of graph searches into systolic arrays;

and

(e) the study of ferroelectric capacitors for application in novel integrated circuits.

Each major section of this report discusses one of these categories.

II. The effect of scaling on the performance of integrated device arrays

The goal was an extension of earlier studies of the fundamental limits of integrated circuits and computational systems [1,2]. A wide variety of possible limiting factors including heat dissipation, signal propagation, energy dissipation in switching operations, clock skew and data stability

have been considered. Our work has attempted to clearly reflect the interaction between these limits in a monolithic integrated circuit. We chose the functional throughput rate (FTR) of a chip to be our performance measure. The FTR is defined as the number of logical operations performed by the chip per unit time and as such includes both technological factors and architectural concerns. This avoids some of the difficulties encountered in measures such as clock speed and also helps us avoid the trap of pursuing schemes for performing dissipationless logic by taking an infinitely long time to perform our elementary functions [3].

We divided the parameters of interest into three classes: (1) fundamental and technological requirements; (2) architectural requirements; and (3) constraints.

Fundamental and technological requirements include minimum power dissipation, maximum speed and minimum dimension or area needed for the elementary processes such as switching, data transfer in space (interconnection) and data transfer in time (storage). Architectural requirements are the number of elementary operations performed by a given computation and architecture. When the fundamental and computing requirements are combined we obtain the total requirements for power dissipation, delay and chip area. Constraints include the heat removal capacity, maximum chip size, maximum I/O bandwidth and clock skew. When the requirements exceed the constraints, limited performance in terms of FTR must result.

We focused on systolic architectures as it is in these architectures that one encounters the worst case heat dissipation condition of having all

the elements in the array performing their function every clock cycle. The usual argument is that the performance of a systolic architecture increases linearly with an increase in the number of processing elements. Arguments for an optimum architecture then are developed by examining a space-time cost function, the total number of processing elements and the number of clock cycles. Such arguments however assume that the maximum clock frequency is independent of the processing element count. Our research effort is directed largely to exploring the conditions under which this last assumption breaks down as a result of interplays between heat dissipation, signal and information propagation and chip area.

We tested some of these ideas by examining a 4-stage, systolic one dimensional convolution architecture implemented using 3 micron CMOS design rules. The analysis of both the 3 micron layout and projected scaled layouts showed that interconnection occupies most of the chip area, even though only connections to nearest neighbors are required for the architectural algorithm. Gate delays, interconnection delays, clock skew , power dissipation and I/O bandwidth interact together to prevent the FTR from scaling in the simple fashion predicted by examination of the architecture and algorithm alone. Furthermore, the factor which limits the performance changes as the system is scaled. For the three micron technology the primary limiting factor is the interconnection delay, while for the scaled technology the primary limiting factor was clock skew.

We then generalized this procedure to a wider range of architectures. We began this task with a study of the class of systolic architectures which perform matrix multiplication. It is known that if one retains the idea of

nearest neighbor interconnects, there are only 27 regular arrays which operate in a systolic fashion while performing matrix multiplication [4]. We therefore chose to examine these 27 systolic architectures from this perspective and to ascertain which, if any, performed matrix multiplication at an enhanced functional throughput level.

An early discovery was that these 27 systolic arrays actually fall into only 4 classes of interconnect patterns. The members of each of these classes differ from other members of the same class of interconnect pattern only in the direction of information flow along the defined paths. To date we have found no physical reason why the direction of information transfer in an architecture should change the fundamental limits on clock frequency. Therefore we need consider only one member of each class to obtain the physical limits on all members of that class. We indeed found that one class appears to be an optimal class of systolic architecture for matrix multiplication.

In this survey we found an interplay between several distinct levels of limiting factors. At the most abstract level one encounters purely problem dependent limits associated with the inherent complexity of a given computational task. At a somewhat less abstract level, there are architectural features associated with the control of data flow in space and time. An architecture however is still an abstract entity which must be physically implemented. When this is done, technological constraints enter in the form of delay and energy dissipation requirements associated with the basic operations. On an integrated circuit all of these are united.

We have since generalized this argument to a far wider class of problems, those which can be performed by regular iterative algorithms. The central point of this work is that a VLSI chip is a vertically integrated information processing system. At the highest level there is a problem statement, something like "multiply two matrices", which is the reason for the chip's existence. The next highest level in the hierarchy is an algorithm, implemented by a hardware structure which in turn is implemented by circuits which in turn are implemented by some defined IC technology e.g. 1 micron CMOS. Our goal is to learn how one can predict whether an improvement in one level of this hierarchy necessarily leads to improved problem solving capability.

The importance of a regular iterative algorithm is that there exists a systematic procedure for describing n-dimensional processor arrays that implement the algorithm. We have developed procedures for directly drawing from the regular iterative algorithm itself the number of time-steps, processor count and I/O pin count. These are the crucial factors in applying technological limits as we did for matrix multiplication. This gives us the ability to derive the overall problem solving capability for any VLSI chip which uses a known technology to implement a regular iterative algorithm for the specified problem. We also have re-examined some of the scaling arguments which have been used to predict ultimate limits to standard VLSI. The limitation of these arguments is that they assume that when a device is scaled we will not make any changes at any other levels of the hierarchy. In our scaling theory all levels of the hierarchy are

changed with device size in order to always provide the optimal problem solving performance for a given device size.

The parameters used in this scaling approach are definite at three levels. The first level are parameters which are specified when the array itself is specified. These parameters include the number of processing elements, the number of I/O words, the processor utilization efficiency (the average number of computations taking place in a PE during a time step), the number of time-steps requires for the computation and the difference in path length along which clocking signals must flow to reach different PE's. The second level is the parameters used to describe the PE itself. These include the number of logic gates, the fraction of these gates that are active in a typical clock cycle, the chip area, the interconnection lengths, worst case paths and fanouts. The last set of parameters needed for our scaling approach is a set of scaling parameters, generally specific to the IC technology being used e.g. CMOS.

In our scaling approach we calculate the FTR by minimizing the product of the timestep itself and the number of timesteps needed for the computation. The number of timesteps required is set by the array design. The timestep length is limited by one of the following four factors: circuit delay, power dissipation, I/O bandwidth and clock synchronization. We compared one and two dimensional arrays for identical problems. We discovered that generally 2-D arrays should be used when circuit delay or power dissipation are limiting factors and that 1-D arrays scale better when performance is being limited by clock synchronization or I/O bandwidth. A detailed discussion of these points can be found in the thesis of Lee [5].

III. The development of a pipelined content addressable memory

In the second major thrust of the program, we were concerned in part with synthetic neural systems, or SNS. This concern however was only in relation to the overall questions concerning integrated circuits, particularly those in the ULSI and WSI regimes. SNS chips are direct attacks on the basic problem of ULSI, which simply put is what do you do with a billion devices anyway? There are three key features that underlie much of the potential success of SNS. These keys are the use of large numbers of PE's, some form of concurrent processing and some form of adaptation. It is important to remember though that neural networks are not the only systems possible that implement these three key features. For example reconfigurable systolic arrays generally have these same features. Other possible architectures can be envisioned as well. We next will review one such architecture, a pipelined content addressable memory, and will use a comparison between it and neural networks as a guide in answering the following question. What are the advantages and disadvantages of SNS when compared with alternative architectures?

The basic structure of this architecture a pipeline, each stage of which contains a word of memory and some comparison logic. Data flows into the top of the pipeline and at the bottom we recover this data along with the address of the pipeline stage which contained in its memory the closest match to the input and a measure of the closeness of this match. Obviously the comparison logic of each stage compares the input data with the stored data, determines the closeness of this match and compares this closeness

with that of the previous best match. The address and closeness of the best of these two is then fed along with the input data to the next stage of the pipeline. A single pipeline has been fabricated and has passed some crude functional testing [6].

At this point we can already begin to compare this scheme with an SNS. First, an SNS usually destroys any input data unless special care is taken whereas here this data is naturally recovered. Secondly, in an SNS functioning as an associative memory no measure of the closeness or goodness of the match between the input "key" and the remembered data is provided whereas here this closeness is supplied. In an SNS interference can occur between two memories and it is often true that when the system learns something new that previous knowledge is degraded somewhat. No such interference or degradation occurs in the pipelined CAM. This interference between individual memories makes it difficult to predict the actual capacity of an SNS while for the pipelined CAM the capacity is easily determined from knowledge of the number of pipelines, the length of these pipelines and the wordlength. While the SNS is resistant to noise degradation of the input data so is the CAM and for largely the same reason. Since the output is chosen by a "best-match" strategy noise degradation of the input introduces errors in the CAM only when it has the effect of corrupting the input in a fashion which causes it to falsely appear to be a different portion of the learned corpus. Such effectively false inputs would fool the SNS as well. For that matter it is difficult to envision any memory which is insensitive to having the input effectively constitute a lie. While an SNS is insensitive to various "soft" errors in the storage

process the error-resistance of the CAM in this regard could be improved quite easily by utilizing the error correcting codes commonly seen in modern memory systems [7]. At this point it would appear that the SNS is an inferior technology but we will now consider an example that shows that while an SNS may in fact be an inferior associative memory that there are other applications in which they should out perform systems built around these pipelined CAMs.

The applications which have been investigated to-date are character recognition and phonetic transcription. The character recognition tests demonstrated that the system is capable of recognizing typewritten characters with accuracy rates and noise tolerance that rival that of existing artificial neural networks. The phonetic transcription problem is of more interest and also is one of the most commonly cited neural network success stories. Sejnowski [8] developed a demonstration program called NET-talk which was taught, using many hours of dedicated VAX time, to "accurately" transcribe English text strings into a recognizable phonetic output. The accuracy of the transcription itself is about 95% and, is often the case with such systems, stress and rhythm are poorly produced as English text contains little information concerning stress and rhythm. We have used a behavioral simulation of 4 parallel pipes in this application and have compared the results with those of NET-talk. We achieved higher accuracies of transcription and recognizable phonetic output (and difficulties with stress and rhythm) after less than an hour of non-dedicated computer time. Less than a minute would be required to teach a real chip the performance of

this task. We have laid out one such pipeline in 3 micron CMOS and it is presently being fabricated by the NSF MOSIS foundry service.

Sejnowski's NETtalk is often viewed as one of the more notable successes of neural networks. It is a software simulation in which a neural net was taught to phonetically transcribe English text. The same task can also be performed by a pipelined CAM [6]. Indeed a pipelined CAM should be quite effective in this task and can rapidly learn to reproduce a large corpus with 100% accuracy. Here however we encounter a feature of an SNS that is very difficult to achieve in another fashion. The tradeoff in performance between the two roughly can be described as giving the learning speed advantage to the pipelined CAM while giving an advantage to the neural net in terms of its ability to respond to novel input, that is words that were not included in the learning corpus. The neural net learns slowly because of the above mentioned interference between individual pieces of knowledge. However, this interference can be constructive and allow the net to identify common features found in the input without having their existence be explicitly described by an external mentor. Therefore it tends to form a more generalized response that allows it to properly respond to input that is novel. For example, after a large number of words have been taught to the neural net, it will tend to properly transcribe the word "hurling" even if it has never seen this word and was taught very similarly spelled words such as "hurting" as part of the learning corpus. The pipelined CAM however under the same circumstances would transcribe the word "hurling" as "hurting" (assuming that this is the closest match from the learning corpus) and therefore make an error. This ability of an SNS to

handle novel inputs in an appropriate fashion is not to be under-rated and in fact may be their most important attribute as it is the most difficult one to emulate with other approaches. The central issue is whether or not it is a direct result of the interference between the various items learned and if so can we alleviate the difficulties posed by this interference without throwing away this advantage as well.

IV. The development of integratable fuzzy neurons

There is another approach which has received some interest which also has many of the same virtues as an SNS. This is to utilize fuzzy logic. A structured approach to the design of learning automatas which use a fuzzy "neuron" as the PE. It appears that these can be implemented by VLSI circuits in a regular PLA like fashion. One goal is to use such elements in the construction of a small chunked neural-net in which the interconnection problem is attacked by decomposing a large neural net into several small modules, each of which has a limited interconnection space. The problem of constructing an acceptor in particular has been addressed but the techniques can be applied to a far wider range of automata [9]. These networks are powerful in computing the acceptance of fuzzy automatas, and thus well suited to tasks like pattern or language recognition.

An acceptor is an automaton which decides whether an input string of symbols belongs to a given language [10]. One limitation of the conventional acceptor is that its output is limited to "accept" or "don't accept". By applying fuzzy theory, a fuzzy automaton [11] which determines

the degree of acceptance for the symbolic inputs can be constructed. These automata are not based on traditional two-valued logic but instead on a logic with fuzzy truth, fuzzy connectives and fuzzy rules of inference [12]. Instead of utilizing a conventional formal language a fuzzy language will be used [13]. A fuzzy language L is a set of ordered pairs, L= (x,u(x)) , where x is a symbolic string and u(x) is the membership grade of x in L. A fuzzy language can be formed in accordance with a set of production rules, in which each production rule is associated with a weight in the closed interval (0,1). As is the case for formal language theory, for any regular fuzzy language there is a corresponding fuzzy finite automaton which can be used to determine the fuzzy acceptance u(x). A fuzzy finite automata, FFA, is a six-tuple FFA = (I,Z,S,M,S_o,T) in which I is a finite input alphabet; Z is an output alphabet; S is a finite set of states; M : S * I * (0,1) -> S is a fuzzy state transition map; S_o which is in S is a vector of initial states; and T, a subset of S, is a vector of final states. The definition is essentially that of non-fuzzy automata with the addition of the fuzzy transition and distributions for the initial and final state vectors.

The fuzzy acceptance can be defined in the following fashion. Let the input string be $x = a_1\ a_2\ ..\ a_n$. We say that the fuzzy automata M accepts x with the fuzzy acceptance d(x), where

$$d(x) = I \circ T(x) \circ F^t$$

$$= I \circ T(a_1) \circ \ldots \circ T(a_n) \circ F^t. \qquad (1)$$

The symbol 'o' denotes the max-min operation; T(a_i) are the fuzzy state transition matrices; and I and F are the vectors of initial and final states respectively. The sequential computation of d(x) is time-consuming. The complexity of computation is further aggravated in the case where a large number of different inputs is involved. However, these difficulties can be eased by exploiting a parallel automata developed using the following mapping procedure.

First, consider the right-hand side of the production rules. If state X is associated with n input alphabets, then X is represented by a set $\{X_i\}$ of n new states. Second, for each production rule, if state X has been expressed by n states then the production rule $X \rightarrow pY$ is expanded to $X_1 \rightarrow pY$, .. , $X_n \rightarrow pY$. This procedure produces a matrix M(x) which describes the interconnection of n neurons where each neuron represents a state X_i. A fuzzy neural acceptor can thus be constructed in accordance with such a matrix. Let x be a_1 .. a_n . The fuzzy acceptance, d(x), can then be expressed as:

$$d(x) = I' \text{ o } M(x) * A^t (a_1) \text{ o } . . . \text{ o } M(x) * A^t (a_n) \text{ o } F'^t \quad (2).$$

where I' and F' are new vectors of initial and final states respectively; A^t (a_i) are activation vectors in which the elements are either 0 or 1 and

a_i is an element of the input alphabet. The symbol '*' denotes an "element-to-element" multiply operation. Note that this multiply operation is not needed in the hardware implementation. The values of expanded states in I' and F' are simply duplicates of the corresponding original states in I and F respectively.

The following simple example will help illustrate the basic concepts of this mapping procedure. Consider the following set of production rules. $A \overset{0.1}{\rightarrow} bB$, $A \overset{0.4}{\rightarrow} aC$, $B \overset{0.3}{\rightarrow} aA$, $B \overset{0.2}{\rightarrow} bB$, $B \overset{0.6}{\rightarrow} aC$ and $C \overset{0.5}{\rightarrow} bA$. The rule $A \overset{0.1}{\rightarrow} bB$ means that if the system is in state A and it is presented with input character b, then it with possibility 0.1 will change to state B. We therefore have two transition matrices in this example, one for input a and the other for input b. These two fuzzy state transition matrices T(a) and T(b) are

$$T(a) = \begin{array}{c|ccc} & A & B & C \\ A & 0 & 0 & 0.4 \\ B & 0.3 & 0 & 0.6 \\ C & 0 & 0 & 0 \end{array} \qquad (3)$$

and

$$\begin{array}{c|ccc} & A & B & C \\ A & 0 & 0.1 & 0 \end{array}$$

$$T(b) = \begin{matrix} B & 0 & 0.2 & 0 \\ C & 0.5 & 0 & 0 \end{matrix} \qquad (4)$$

Here the difficulty is that we have two possible interconnection patterns. The procedure described above however produces the following single interconnection matrix M(x) is

	A_a	A_b	B_{b1}	B_{b2}	C_{a1}	C_{a2}
A_a	0	0	0.1	0	0.4	0
A_b	0	0	0.1	0	0.4	0
B_{b1}	0.3	0	0	0.2	0	0.6
B_{b2}	0.3	0	0	0.2	0	0.6
C_{a1}	0	0.5	0	0	0	0
Ca2	0	0.5	0	0	0	0

Note that the zero elements in T(a) and T(b) denote the "non-membership" of the state transitions, a concept which has no direct hardware interpretation while the zero elements in the matrix M(x) denote the lack of a hardware interconnection between the elements which represent each of the two possible expanded states. This will be a fuzzy neuron whose output is equal to a number r if it is firing and s if it is not firing, where $0 < r , s < 1$, and is computed using the above max-min operations. This PE will consist of two parts, a performance branch and a learning or adaptation branch. A

performance branch contains a minimum unit which performs the fuzzy intersection operation; a maximum unit which performs the fuzzy union operation; registers R1, W, and R2 which hold an internal state value, a weight value (the non-zero value in the associated column of the M(x) represents the weight), and an intermediate result respectively; and a control block which determines the firing status. The learning branch is capable of updating the values in registers W and R1.

It is the performance element of this fuzzy neuron (FN) that corresponds to the McCulloch-Pitts neuron (MPN). The FN intersection - minimum operation corresponds to the MPN weighted multiplication, the FN union-maximum operation to the MPN summation and the FN trigger-activation to the MPN thresholding. The conventional MPN contains no learning or adaptation behaviors of its own and instead is totally reliant on external processors for the performance of this function.

The system functions in the following fashion. To begin, the initial state and weight value of each neuron are loaded in registers R1 and W respectively. Then all the neurons execute concurrently the same sequence of instructions in the following time steps. The neurons will be fired if the corresponding type of input is received. The operation $I' \circ M(x) * A^t (a_i)$ is performed by having the neurons (in parallel fashion) first load register R2 with the min of r1 (the initial contents of register R1) and w (the weight stored in register W). Then the maximum of the various inputs I_i is found. If the neuron is firing, that is if the appropriate input (a or b in our particular example above) has been presented to the overall

network, then this maximum is stored in register R1 and constitutes the new state for the neuron. If the neuron is not firing then 0 is stored in R1.

Simple and regular data flows are the primary concern of VLSI system design. However, the interconnection in itself here will not be regular if a straight forward variation on the processing element is utilized. One approach to overcome these interconnection difficulties is to decompose the fuzzy neurons into basic functional blocks. By this functional decomposition approach, all of the minimum and maximum units of a fuzzy acceptor are combined into a MIN plane and a MAX plane respectively. All the weight and state registers are accordingly combined into two register files. Note that if the weights in the productions are either 1 or 0 then the minimum and maximum operations can simply be realized by AND and OR gates respectively. Therefore, the structure of a fuzzy acceptor is reduced to a PLA form.

V. The mapping of graph searches into systolic arrays

A variety of models useful in artificial intelligence utilize labeled or unlabeled directed graphs. By extending the concepts discussed in the previous section we developed procedures by which basic graph search operations could be directly mapped into an integrated circuit. This id done by first replacing the graph by a Boolean matrix-vector product.

The vector has a length equal to the number of vertices in the graph. Each element of the vector is a simple Boolean variable which denotes whether or not this particular vertex is activated or reached in the search.

The matrix is a square matrix of identical side length. Its elements are simple Boolean variables which denote whether or not an edge of the graph extends from vertex i to vertex j. It is a Boolean form of the adjacency matrix of the graph. For labeled graphs, it is necessary to keep track of the labeling of these edges. Techniques for doing this were developed.

The graph search then can be performed by repeated application of the product

$$R_{n+1} = M ** R_n \tag{5}$$

where R_k is the vector at the kth iteration, M is the adjacency matrix and ** denotes a Boolean inner product in which we form the jth element of R_{n+1} by ORing a set of terms in which the ith term is formed by ANDing the ith element of R_n and the matrix element M_{ji}.

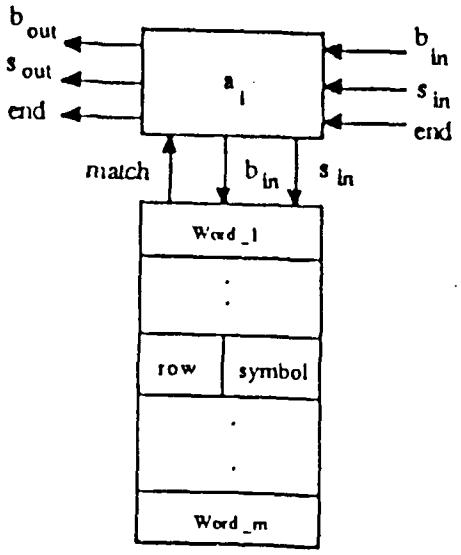
The complicating feature of course is the labeling of the edges. The most direct process for handling labeled graphs is to use a set of matrices, each of which is applied for a given label. The labels are then viewed as externally supplied control information. We first describe this particular approach. We will then describe a second technique.

A rich literature exists for systolic algorithms for arithmetic matrix operations [14]. The above rule is identical to the arithmetic matrix-vector product in terms of data and control flow and therefore we could apply such standard systolic techniques to this problem. However, a common feature of the matrix M is that it will be sparse. We have developed a

systolic array, similar to others intended for sparse matrices [15], which handles both the sparcity and the existence of labels as well. This array is shown in figure 1. Each array element consists of a very simple logic unit and an associated column of content addressable memory. There is a one-to-one mapping of vertices onto these array elements. The operations of the array are described in figure 1 as well. Note that it handles labels by storing the edges and their labels in the content addressable memory columns. The addressing of this column therefore essentially picks out the appropriate M matrix.

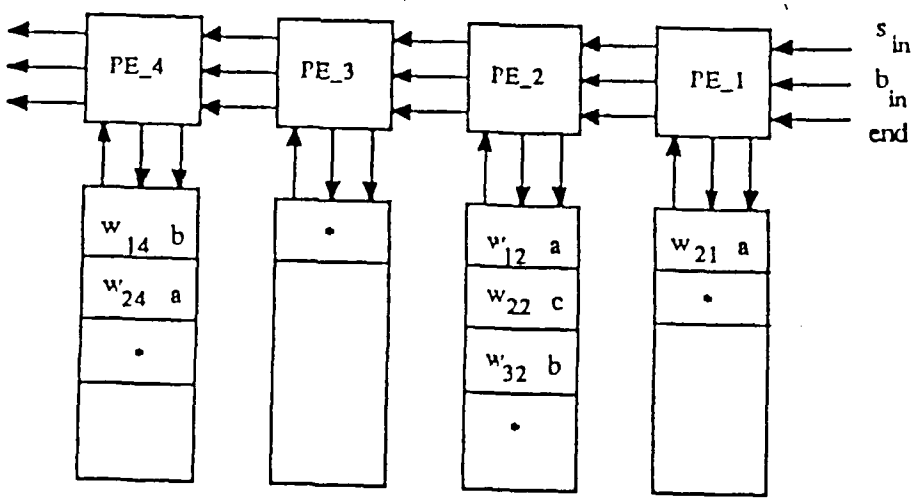
A second procedure was developed for handling labeled graphs. It proceeds by realizing that we can divide labeled directed graphs into two general categories. One of these categories can be treated by applying a systolic array similar to that of figure 1 except for the fashion in which labels are used.. Any directed graph that is not a member of this category can be expanded into a larger equivalent graph which is a member of this category.

The two categories are uniformly and non-uniformly labeled directed graphs. In a uniformly labeled graph, each and every vertex is reached only by edges with identical labels. In a non-uniformly labeled graph, there is at least one vertex which is reached by edges with different labels. For a uniformly labeled directed graph, each row of the adjacency matrix is associated with only one label. Such graphs can be searched by the systolic array shown in figure 2. This array is similar to that of figure 1. Again, each vertex is represented by a single array element and each array element consists of a very simple logic unit and a column of content addressable



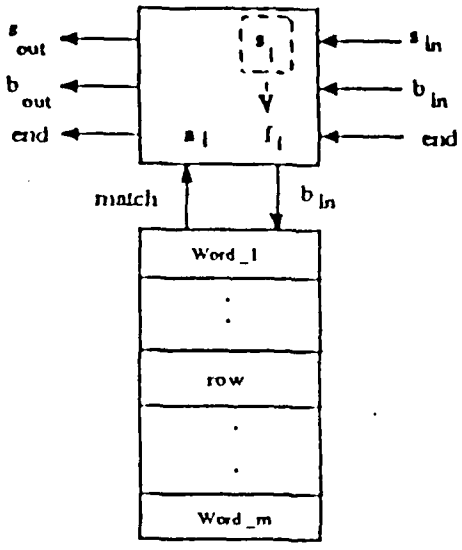
$$a_i \leftarrow a_i \vee \{index(b_{in}) \otimes row\_index(w_{ij})\} \wedge \{index(s_{in}) \otimes index(symbol)\}$$

(a) Processing element and its column of associative memory



(b) Systolic array

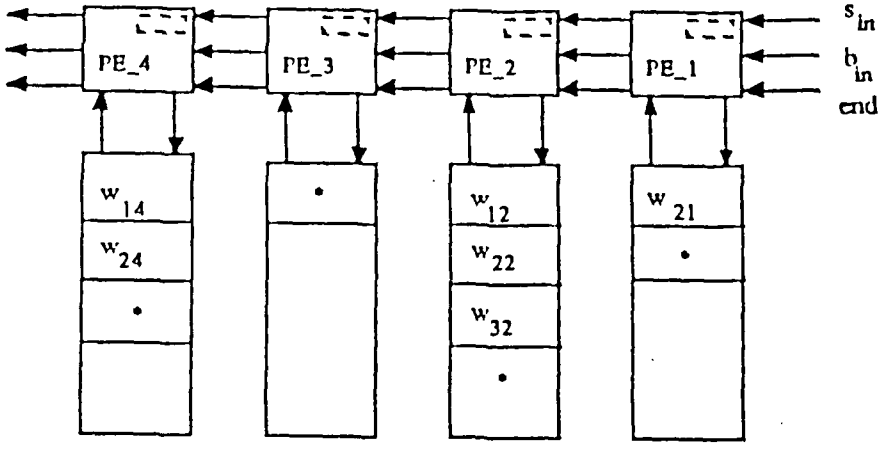
Figure 1. Systolic Array for Labeled Directional Graph Searches.



$$f_i \leftarrow \{index(s_{in}) \otimes index(s_i)\}$$

$$a_i \leftarrow a_i \vee \{index(b_{in}) \otimes row\_index(w_{ij})\} \wedge f_i$$

(a) Processing element and its column of associative memory



(b) Systolic array

Figure 2. Systolic Array for Uniform Labeled Directional Graph Searches.

memory. The difference is in how labels are handled. Since each node is uniformly labeled, we no longer need to store labels in the content addressable memory column. However, in the logic unit the label s_{in} is compared with a stored label s_i. If these two do not match, the vertex is not reached. Nonuniformly labeled graphs can be transformed into a larger. uniformly labeled graph by a technique of vertex splitting. Any vertex which is reached by edges with more than one label is split into a set of vertices, each of which is reached uniformly labeled edges. In this splitting procedure, any edge which was directed away from the original vertex is simply reproduced for each of the new vertices.

An unsupported graduate student is presently laying out a test chip containing such a systolic array. We believe that these arrays will be well suited for problems in which we need to repeatedly and rapidly search the same sparsely connected graph, as we have essentially replaced a software search by a direct hardware implementation.

VI. The study of ferroelectric capacitors for application in novel integrated circuits.

A primary target of an integrated circuit implementation of the more "traditional" analog forms of a SNS is a good "synapse". The "synapse' is a weighted connection between two neurons. A good one has a continuous range of analog weight values available. These values must be easily changes by electronic signals available in the basic IC technology and yet must also be

relatively stable. We want its value only to change when we want it to change. Additionally, we also desire that the synapse have the normal virtues of small chip area and low power consumption. We have investigated the potential of ferroelectric elements for such a synapse. This effort is certainly not restricted to SNS applications however. In the process of considering various circuits, we have encountered an important practical problem: the lack of a good circuit model in the standard public domain integrated circuit simulation packages makes it difficult to simulate circuit concepts. As a part of our effort we have begun the development of such a model, an effort which should be useful in the study of other applications of such ferroelectric elements on ICs.

Two different synaptic connection circuits have been studied. The first uses a straight-forward extension of techniques used in ferroelectric RAM's [16] to provide a binary connection weight. In the second, a novel analog memory element is used to provide a continuous valued, analog weight. Unlike the existing ferroelectric RAM, this novel element uses a nondestructive read-out.

The binary circuit is shown in figure 3. The neuron consists of a sense amplifier, whose positive and negative inputs are connected to ferroelectric capacitors which represent exhitory and inhibitory synapses. The operation starts with the application of a pulse signal. Capacitors whose polarization is aligned with the pulse field are off connections while those whose polarizations oppose the pulse electric field are "on" connections. When the axon lines are pulses, differing amounts of displacement current are injected by capacitors in these two different

polarization states. In this process, synapse capacitors which were initially "on" are switched into the other polarization state hence the phrase "destructive read out". This circuit is susceptible to an associated material fatigue limit on the number of occasions in which the film can be switched.

The continuously valued weight circuit is shown in fig. 3. It uses a Sawyer-Tower circuit to bias a subthreshold transistor, which acts as a voltage controlled current source. In this system we use the ferrolectric capacitor to supply this controlling voltage. This is done by adjusting the voltage V_x. Various modes of operation are possible which use either partial polarization switching or minor hysterisis loops [17].

In order to simulate such circuits we have developed a model for a ferroelectric capacitor which is implemented in SPICE. (This particular project was started late in the program reported here and was finished as part of a second DoD supported activity in neural networks.) In SPICE, modified nodal analysis is used to represent all circuit branches in terms of elements which are voltage defined and current controlled e.g. conductances and current sources. Such models are essentially DC only but SPICE performs time varying problems by concatenating a series of such solutions at each time step. Nonlinear elements are linearized in a Newton-Raphson-like procedure and energy storage elements, such as capacitors, are described using a trapezoidal integration procedure. In these last two case, those of nonlinear elements and energy storage elements, the

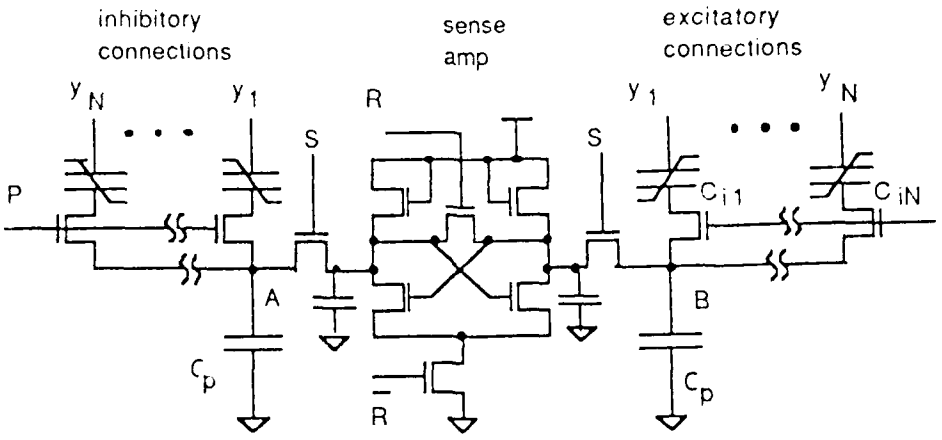


Figure 3 Neural circuit supporting binary weights.

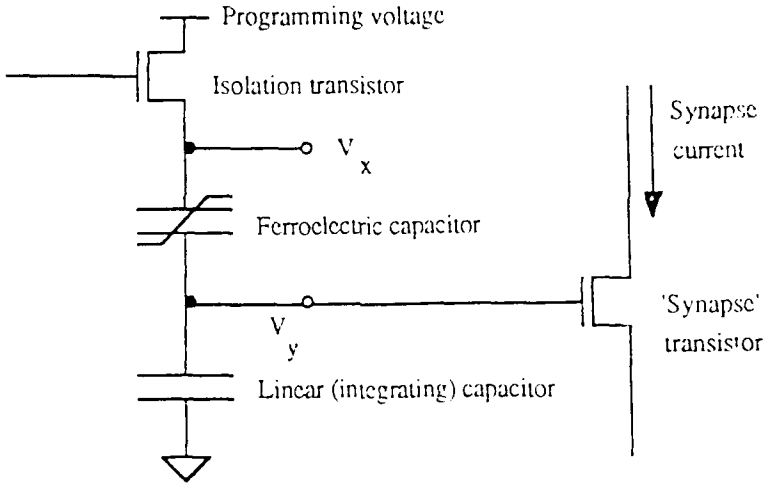


Figure 4 Analog synapse circuit. The synapse transistor may be replaced by any transconductance amplifier, e.g., for greater multiplication linearity.

analytical techniques described are implemented in the circuit analysis by an equivalent circuit formed from conductances and current sources. The ferroelectric elements of concern to us are nonlinear capacitances. Simply adjusting the capacitance of the basic SPICE linear capacitor model does not include the nonlinearity. The simplest form of a nonlinear model is to use a hysterisis curve. However, the hysterisis curve is merely the dc steady-state solution to the polarization transient problem and we are interested in these transients. We therefore have chosen to use a set of rate equations [18] whose solution describes the number of dipoles oriented with the field as a function of time. The resulting model includes a basic linear capacitor, a nonlinear ferroelectric switching current modeled by application of the Newton-Raphson technique to the rate equations, and a parallel conductance which represents any leakage current through the film. The model gives rise to proper I-V, hysterisis loop and current vs. time characteristics.

VII. References

1. W. Porod, R.O. Grondin, D.K. Ferry and G. Porod, Physical Review Letters, 52, 232 (1984)

2. R.O. Grondin, W. Porod and D.K. Ferry, IEEE J. Solid-State Circuits, SC-19, 262 (1984)

3. R. Landauer, IBM J. Res. Dev. 5, 183 (1961)

4. S.K. Rao, Regular Iterative Algorithms and Their Implementations on Processor Arrays, Ph.D. dissertation, Stanford University, October 1985

5. Hoon Bock Lee, Performance of VLSI Processor Arrays for Regular Iterative Algorithms, Ph.D. dissertation, Arizona State University, 1988

6. L.T. Clark and R.O. Grondin, Proc. First IEEE Neural Nets Conf., San Diego, pp. III-411, 1987

7. C.L. Chen and M.Y. Hsiao, IBM. J. Res. Dev., 28, 124 (1984).

8. T.J. Sejnowski and C.R. Rosenberg, NETtalk: A Paallel Network that Learns to Read Aloud, tech. report JHU/EECS-86-01, the Johns Hopkins University, EE and CS tech reports, (1986)

9. L.C. Shiue and R.O. Grondin, Proc. First IEEE Neural Nets Conf., San Diego, pp. II-299, 1987

10. D. Hopkin and B. Moss, Automata, North-Holland, New York, 1976.

11. W.G. Wee and K.S. Fu, IEEE J. Solid State Circuits, SSC-5, no. 3, (1969).

12. L.A. Zadeh, Journal of Cybernetics, 2, 4 (1972).

13. E.T. Lee and L.A. Zadeh, Information Sciences 1 421 (1969).

14. S.Y. Kung, VLSI Array Processors, Prentice Hall, Englewood Cliffs, 1988

15. O. Wing, J. Parallel Distributed Computing, 2, 170 (1985)

16. S. Eaton, D. Butler, M. Parris, D. Wilson and H. McNeillie, IEEE Solid State Circuits COnf. Digest of Technical Papers, p. 130. 1988

17. Lawrence T. Clark, Robert O. Grondin and Sandwip K. Dey, First IEE Conf. on Artificial Neural Networks, London, England, pp. 47-51, IEE Conference Publication No.313, October 1989

18. M. Cabezuelo, J. Lorenzo and J. Gonzalo, Ferroelectrics, 87, 353 (1988)

OFFICE OF NAVAL RESEARCH

PUBLICATIONS / PATENTS /PRESENTATIONS/ HONORS REPORT

for

1 October 1985 through 30 September 1989

for

Contract N00014-85-K-0387

Basic Properties and Limits of Integrated Arrays
of Dissipative Circuit and Logic Elements

Robert O. Grondin

Arizona State University

Tempe, AZ 85287

Book Chapters

1. W. Porod, R.O. Grondin and D.K. Ferry, "Interconnection, Dissipation and Computation," in VLSI Electronics, Vol. 15, Ed. N. Einspruch, S.S. Cohen and G. Gildenblat, Academic Press, Orlando, Fl, 1987.

2. D.K. Ferry, R.O. Grondin and L.A. Akers, "Two Dimensional Automata in VLSI," in Submicron Integrated Circuits, R.K. Watts, Ed., John Wiley, New York, 1989.

3. L.A. Akers, R.O. Grondin and D.K. Ferry, "Synthetic Neural Systems," for An Introduction to Neural and Electronic Networks, Ed. by S.F. Zornetzer, J.L. Davis and C. Lau, Academic Press, in press

4. D.K. Ferry, L.C. Shiue, R.O. Grondin and L.A. Akers, in Frontiers in Computing Systems Research, Ed. S.K. Tewksbury, Plenum Annual Book Review Series

Doctoral Dissertations

1.	Hoon Bock Lee,	Performance of VLSI Processor Arrays for Regular Iterative Algorithms, Ph.D. Electrical Engineering, June 7, 1988
2.	Liang Chyi Shiue	SCALX: A VLSI Architecture for Concurrent Symbolic ProcessingPh.D. Electrical Engineering, December 1989
3.	Lawrence T. Clark	work in progress, Ph.D. Electrical Engineering, expected completion date 1991

Masters Theses:

1. L. T. Clark, "A Novel VLSI Architecture for Cognitive Applications,", masters thesis, Department of Electrical & Computer Engineering, Arizona State University, January 1987

Refereed Journal publications:

1. H.B. Lee and R.O. Grondin,"A Comparison of Systolic Architectures for Matrix Multiplication," IEEE J. Solid State Circuits, Vol. Sc-23, No. 1, pp. 285-289, 1988

2. L.T. Clark and R.O. Grondin, "A Pipelined Associative Memory Implemented in VLSI," IEEE J. Solid State Circuits, Vol. 24, No. 1, pp. 28-34, February 1989

Papers Submitted to Refereed Journals, in review:

1. L.T. Clark, S.K. Dey and R.O. Grondin, "Ferroelectric Thin-Film Memory for Electrically Programmable IC Neural Networks," submitted to Ferroelectrics

Conference Papers:

1. D.K. Ferry, J.M. Golio and R.O. Grondin, "Interconnections and Limitations in VLSI," 1985 Proc. Second Inter. IEEE VLSI Multilevel Interconnection Conf., Santa Clara, CA, pp. 408-415

2. L.T. Clark and R.O. Grondin, "Comparison of a Pipelined 'Best Match' Content Addressable Memory with Neural Networks," IEEE First International Conference on Neural Networks, San Diego, CA. pp. III-411-418, proceedings published by the IEEE, 1987

3. L.C. Shiue and R.O. Grondin, "On Designing Fuzzy Learning Neural-Automata," IEEE First International Conference on Neural Networks, San Diego, CA. pp. II-299-307, proceedings published by the IEEE 1987

4. L.C. Shiue and R.O. Grondin, "An Automata Approach to Reconfigurable Learning-Network Design," in Neural Networks from Models to Applications, pp. 380-388, Ed. by L. Personnaz and G. Dreyfus, I.D.S.E.T., Paris, 1989

5. L.C. Shiue and R.O. Grondin, "Neural Processing of Semantic Networks," presented at the Inter. Joint Conf. on Neural Networks, Washington DC, abstract published on pg. II-598 of proceedings, June 1989.

6. Lawrence T. Clark, Robert O. Grondin and Sandwip K. Dey, "IC Neural Networks with Ferroelectric Capacitor Connections", First IEE Conf. on Artificial Neural Networks, London, England, pp. 47-51, IEE Conference Publication No.313, October 1989

Abstracts submitted to conferences, in review:

1. L. T. Clark, R.O. Grondin and S.K. Dey, "Electrically Programmable Analog Synapses Using Ferroelectric Thin-Film Memory," submitted to NIPS 1990

Patents

Pipelined Best Match Content Addressable memory, Lawrence T. Clark

Graduate Students Supported

Hoon Bock Lee (Ph.D)
Liang Chyi Shiue (Ph.D)
Lawrence T. Clark (MS and Ph.D)

Undergraduate Students Supported

Kevin Connolly, summer 1989